Visualizing airborne CO_2 data

Prepared by the ORNL DAAC https://daac.ornl.gov (https://daac.ornl.gov)

August 16, 2022

Introduction

This exercise will explore properties of in-situ measurements of atmospheric gases from airborne platforms and illustrate how to visualize an example data set as well as subset it by temporal and spatial bounds. It will use data from *ABoVE_2017_insitu_10sec.nc*, a file downloaded from Sweeney and McKain (2019) (https://doi.org/10.3334/ORNLDAAC/1658). Please review the The User Guide (https://daac.ornl.gov/ABOVE/guides/ABoVE_Arctic_CAP.html) for this data set before proceeding. This guide contains important information on the organization of the data set and variables. It will be helpful to have the User Guide open for quick reference.

The data set includes measurements of carbon dioxide (CO_2), methane (CH_4), and carbon monoxide (CO) concentrations taken during flights over Alaska, Canada, and the continental U.S. in 2017. During the flights, air samples were collected and analyzed using on-board instrumentation. Each data point is a time-stamped measurement taken in 3-dimensional space (longtitude, latitude, altitude). During 2017, the flight lines (Figure 1) were repeated several times between April to November, with some spatial variability, in order to sample sample carbon dynamics over northern biomes during the growing season.

Artic-CAP flight paths 2017

Figure 1. General path of flight lines for the Artic-CAP project for 2017. Flight lines were repeated several times during the growing season. Sources: Sweeney and McKain (2019); Scientific Aviation (2019)

This file includes approximately 45 variables, but all of them are not needed for this exercise. We will use seven variables listed in Table 1 and focus on the CO_2 measurements. These variables are indexed by a single dimension.

Table 1. Variables used in this exercise.

Variable	Units/format	Description
altitude	m.a.s.l.	Sample altitude (GPS altitude) in meters above sea level
flight_id	YYYYMMDD	A unique number identifying each flight. The format is the date in YYYYMMDD on which the flight began. See Table 1.
CO2	umol per mol	Mole fraction of carbon dioxide in dry air; average of all measurements made in the time interval. Mole fraction reported in units of micromole per mole (1e-6 mol per mol of dry air); equivalent to ppm (parts per million). Fill value: -9999
CO2_unc	umol per mol	Estimated uncertainty of the reported value. May be a single average uncertainty value for the whole data set. The mole fraction reported in units of micromole per mole (1e-6 mol per mol of dry air); equivalent to ppm (parts per million).Fill value: -9999
latitude	Decimal degrees	Latitude at which air sample was collected
longitude	Decimal degrees	Longitude at which air sample was collected
time	seconds since 1970-01-01T00:00:00Z	Number of seconds since January 1, 1970 in UTC. Time-averaged values are reported at the beginning of the averaging interval.

The rest of the document illustrates how visualize the data on a map and examine patterns in CO_2 concentrations by altitude, latitude, and longitude. Methods for creating subsets of the data by time and space are included.

Install and load R packages

The R packages employed in the exercise are:

- RNetCDF - for opening and reading NetCDFs
- lubridate - for manipulating dates and times
- ggplot2 - for creating graphs
- sp - for processing coordinates and spatial information
- scales - for making pretty axes on figures
- tmap and tmaptools - for displaying maps of flight paths
- OpenStreetMap - for downloading a base map

```
# Use this code to install packages as needed
install.packages("RNetCDF", dependencies = TRUE)
install.packages("lubridate", dependencies = TRUE)
install.packages("ggplot2", dependencies = TRUE)
install.packages("sp", dependencies = TRUE)
install.packages("scales", dependencies = TRUE)
install.packages("tmap", dependencies = TRUE)
install.packages("tmaptools", dependencies = TRUE)
install.packages("OpenStreetMap", dependencies = TRUE)
```

```
library(RNetCDF)
library(lubridate)
library(ggplot2)
library(sp)
library(scales)
library(tmap)
library(tmaptools)
library(OpenStreetMap)
```

Load data

This R code opens the NetCDF for reading metadata and variables.

```
nc <- open.nc("ABoVE_2017_insitu_10sec.nc")   # NetCDF file must be in the working directory
  # print.nc(nc)    # Use to display metadata about variables and structure; same as NCO 'ncdump -h'
```

All 45 variables are not needed, so build a dataframe, *d*, by reading one variable at a time (Table 1).

```
# read in one variable at time to build a dataframe named 'd'
  x <- var.get.nc(nc, "time")
  d <- data.frame(time=x)
  d$lat <- var.get.nc(nc, "latitude")
  d$lon <- var.get.nc(nc, "longitude")
  d$altitude <- var.get.nc(nc, "altitude")
  d$co2 <- var.get.nc(nc, "CO2")
  d$co2_unc <- var.get.nc(nc, "CO2_unc")
  d$flight_id <- var.get.nc(nc, "flight_id")
close.nc(nc)    # closes the NetCDF
```

Records without CO_2 data should be deleted, resulting in a dataframe with 95,415 records.

```
d <- d[!is.na(d$co2), ]   # delete records without CO2 data
d.bak <- d                # back up dataframe
dim(d)                    # 95415     7
```

```
## [1] 95415     7
```

Visualize the spatial extent

First, let's visualize the spatial extent of the data. The code below creates a spatial lines object from the dataframe then displays it on a map. Note that downloading the base map from OpenStreetMaps (*read_osm()*) may be slow depending on internet speeds.

```
## Plot the flight paths for entire data set
gcs <- CRS("+proj=longlat +datum=WGS84")  # projection information for lon lat coordinates
ID <- paste0("line_",1:100)
spl.0 <- Line(cbind(d$lon, d$lat))        # create a line object from dataframe
spl.1 <- Lines(list(spl.0), ID=ID[1])
spl <- SpatialLines(list(spl.1), proj4string = gcs)
rm(spl.0, spl.1)                          # clean up

# create map with background from OpenStreetMaps
map <- read_osm( bb(spl), ext=1.1 )       # the background map; use bbox from spl
tm_shape(map) + tm_rgb() + tm_shape(spl) +
    tm_lines(col= "blue", lwd= 3) + tm_graticules()  +
  tm_credits("Figure 2. Paths of all flights in data set.", align="left", size=1.0) +
  tm_layout(attr.outside = T, attr.position=c("left", "bottom"), attr.outside.position = "bottom")
```

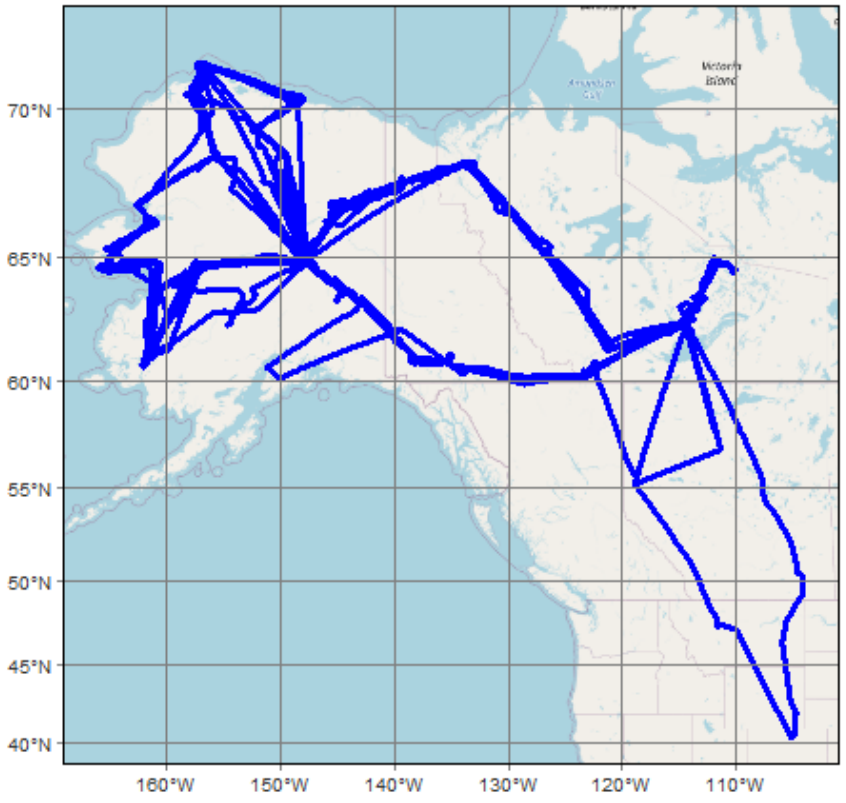

Figure 2. Paths of all flights in data set.

Subsetting the data by dates

It is often necessary to select data for particular dates or locations. Date and time of measurement are recorded in the *time* variable. Units for *time* are “seconds since 1970-01-01 00:00:00”. This code uses utilities from the R lubridate package to convert *time* values to dates, then prints a list of the dates.

```
# Print list of dates
origin <- as.Date("1970-01-01 00:00:00")              # starting pt for count of seconds (Table 1)
(origin + seconds(d$time)) %>% date() %>% unique()
```

```
##  [1] "2017-04-26" "2017-04-27" "2017-04-28" "2017-04-29" "2017-04-30"
##  [6] "2017-05-01" "2017-05-02" "2017-05-03" "2017-05-04" "2017-06-06"
## [11] "2017-06-07" "2017-06-08" "2017-06-09" "2017-06-10" "2017-06-13"
## [16] "2017-06-14" "2017-06-18" "2017-06-19" "2017-07-09" "2017-07-10"
## [21] "2017-07-12" "2017-07-13" "2017-07-14" "2017-07-15" "2017-07-17"
## [26] "2017-07-18" "2017-07-19" "2017-07-20" "2017-07-21" "2017-08-17"
## [31] "2017-08-18" "2017-08-21" "2017-08-22" "2017-08-23" "2017-08-24"
## [36] "2017-08-26" "2017-08-27" "2017-08-28" "2017-08-29" "2017-08-30"
## [41] "2017-09-08" "2017-09-09" "2017-09-10" "2017-09-13" "2017-09-15"
## [46] "2017-09-17" "2017-09-18" "2017-09-19" "2017-09-21" "2017-09-22"
## [51] "2017-09-24" "2017-09-27" "2017-09-28" "2017-10-18" "2017-10-21"
## [56] "2017-10-22" "2017-10-23" "2017-10-24" "2017-10-25" "2017-10-31"
## [61] "2017-11-01" "2017-11-02" "2017-11-04" "2017-11-05"
```

These functions will help in subsetting by date & time, geographic location, or flight ID number.

```
# function to filter by date-time
time.filter <- function(df,tstrt,tend){
  ### pass dataframe with starting & ending times in "%Y-%m-%d" UTC (e.g., "2017-04-26")
  origin <- as.Date("1970-01-01 00:00:00", tz = "UTC")
  time.start <- paste0(tstrt," 00:00:00") %>% strptime(., "%Y-%m-%d %H:%M:%S", tz = "UTC")
  time.end   <-  paste0(tend," 23:59:59") %>% strptime(tend, "%Y-%m-%d %H:%M:%S", tz = "UTC")
  t.strt <- difftime(time.start, origin, units="sec")
  t.end <- difftime(time.end, origin, units="sec")
  t.ind <- which((df$time>=t.strt & df$time<=t.end), arr.ind=T) # index
  return(df[t.ind,])        # return records within date range
                            }   # end of time.filter func

# ... by geographic coordinates
coord.filter <- function(df, xmin, xmax, ymin, ymax){ # the passed dataframe must have 'lon' and 'la
t' fields
   t.i <- which(df$lat<ymax & df$lat>ymin &
                df$lon<xmax & df$lon>xmin)
  return(df[t.i,])      # return records within date range
                            }   # end of coord.filter func

# ... by flight_id number
flight.filter <- function(df, flt){ # flt = 8-digit flight_id
  df <- df[df$flight_id == flt,]
  return(df)        # return records within date range
                            }   # end of flight.filter func
```

Before proceeding, make a backup of the dataframe *d* because the code below will alter it.

```
d.bak <- d
# d <- d.bak  # When needed to revert to the full dataframe.
```

Let's select a specific flight from 9 July 2017 (2017-07-09) using its *flight_id* (2017-07-09). The *flight_id* values are listed in Table 1 of the User Guide, and they correspond to the date of the flight in YYYY-MM-DD without the "-". For example, the date "2017-07-09" corresponds flight_id "20170709" .

```
d <- flight.filter(d, 20170709) # replaces dataframe with filtered data
dim(d) # 1246    7
```

```
## [1] 1246    7
```

The map for this flight is drawn using the same code used above (code chunk: “map-all-flights”).

```
spl.0 <- Line(cbind(d$lon, d$lat))        # create a line object from dataframe
spl.1 <- Lines(list(spl.0), ID=ID[1])
spl <- SpatialLines(list(spl.1), proj4string = gcs)
rm(spl.0, spl.1)                          # clean up
map <- read_osm( bb(spl), ext=1.1 )       # uses bbox from spl
tm_shape(map) + tm_rgb() + tm_shape(spl) +
    tm_lines(col= "blue", lwd= 3) + tm_graticules(labels.cardinal=F) + tm_layout(asp=0.75) +
  tm_credits("Figure 3. Path of flight 20170709", align="left", size=1.0) +
  tm_layout(attr.outside = T, attr.position=c("left", "bottom"), attr.outside.position = "bottom")
```

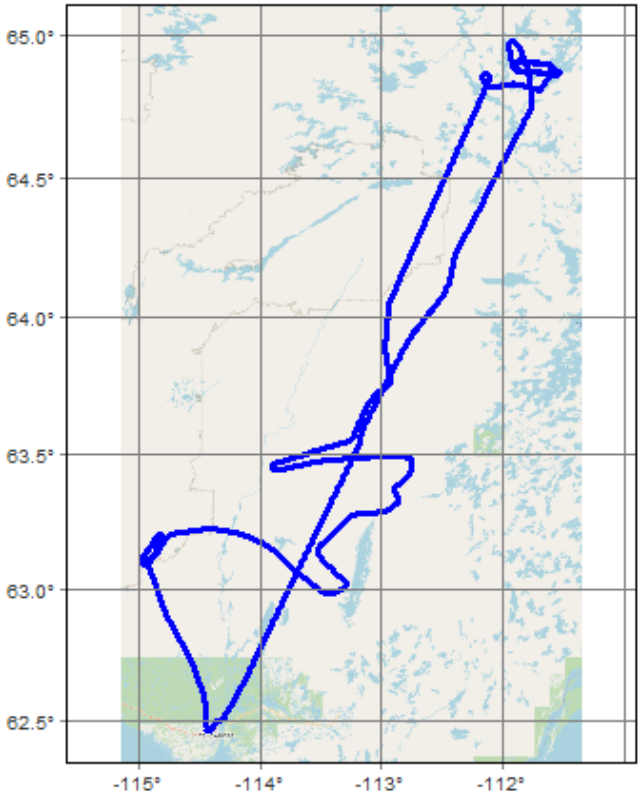

Figure 3. Path of flight 20170709

The flight starts and end at an airfield near longitude -114.5, latitude 62.4.

Examine CO_2 concentrations

To see how CO_2 varied with aircraft altitude, plot altitude against time and color points by CO_2 concentration.

```
# Plot CO2 concentrations by altitude
t.lab <- origin + seconds(d$time)    # time label for x axis
# create dataframe 'pdat' for plotting
pdat <- data.frame(time.lab=t.lab, time=d$time, alt=d$altitude,
        lat=d$lat, lon=d$lon, co2=d$co2)
ggplot(pdat, aes(x=time.lab,y=alt)) + geom_point(aes(color=co2)) +
    scale_color_gradientn(colors = c("blue", "green", "orange", "red")) +
    scale_x_datetime(breaks = scales::date_breaks("30 mins"),     # adjust date_breaks as needed for
 x axis label
        date_labels = "%H:%M") +    theme_classic()  +
    labs(x="Time (UTC)", y="Altitude (m)", caption="Figure 4.  CO2 concentrations by altitude and tim
e.") +
  theme(legend.position="bottom", plot.caption=element_text(hjust=0, size=12))
```

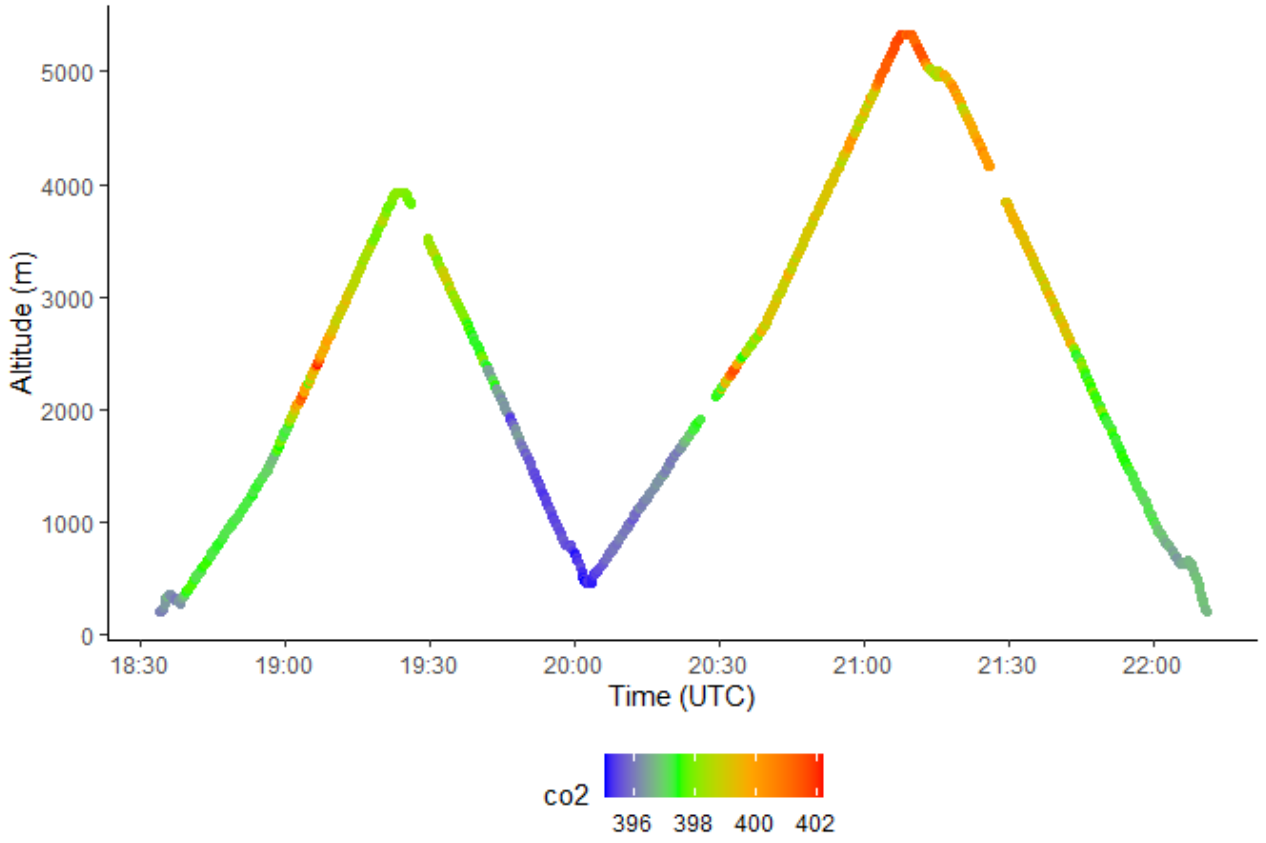

Figure 4. CO2 concentrations by altitude and time.

Plotting by latitude or longitude can be useful. Use latitude here because this flight has a primarily north-south route (Figure 3).

```
# Plot CO2 concentrations by altitude
t.lab <- origin + seconds(d$time)    # time label for x axis
# create dataframe 'pdat' for plotting
pdat <- data.frame(time.lab=t.lab, time=d$time, alt=d$altitude,
        lat=d$lat, lon=d$lon, co2=d$co2)
ggplot(pdat, aes(x=lat,y=alt)) + geom_point(aes(color=co2)) +
    scale_color_gradientn(colors = c("blue", "green", "orange", "red")) +
    theme_classic()  +
    labs(x="Latitude", y="Altitude (m)", caption="Figure 5. CO2 concentrations by altitude and latitu
de.") +
  theme(legend.position="bottom", plot.caption=element_text(hjust=0, size=12))
```

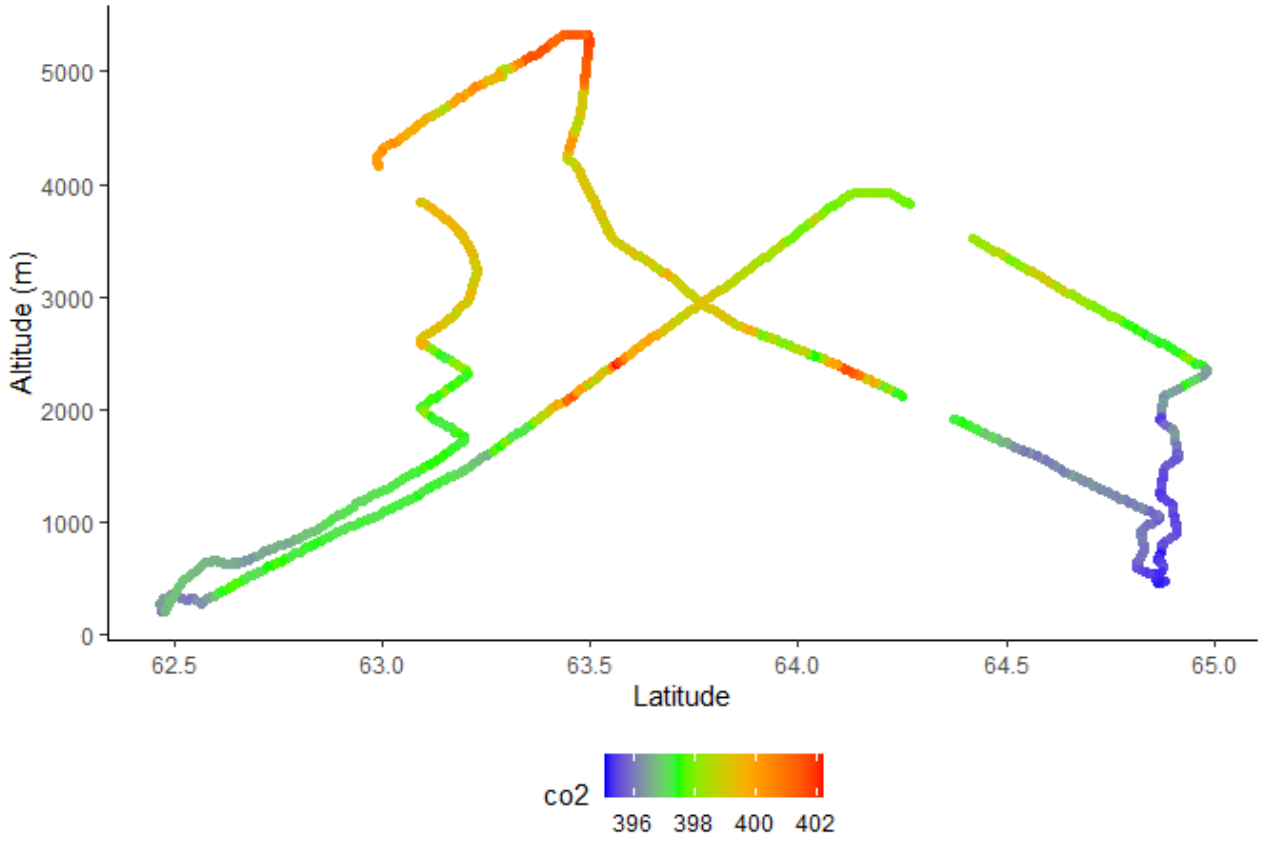

Figure 5. CO2 concentrations by altitude and latitude.

Figures 3-5 show that this flight flew north first with a gradual increase in altitude. At its northern extent, it took a spiraling descent to sample a vertical profile of atmospheric chemistry. On the way back south, the aircraft ascended to >5000 m before executing another spiraling descent around 63.1 degrees latitude.

Zooming in by coordinates

The spiral flight in the northwest corner of Figure 3 might be interesting. Let's zoom into that area using coordinates.

```
# Specifying a bounding box
lon.min <- -112.0   # western longitude
lon.max <- -111.0   # eastern longitude
lat.min <- 64.5   # southern latitude
lat.max <- 65.0   # northern latitude
d <- coord.filter(d, lon.min,lon.max,lat.min,lat.max)
```

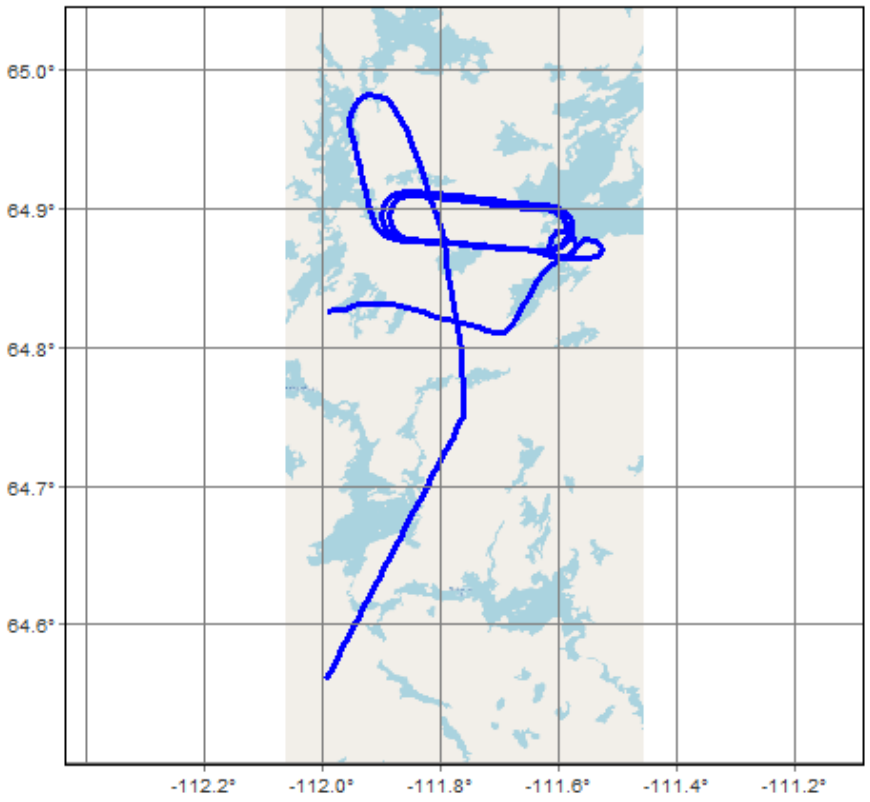

Figure 6. Northeast portion of flight path from Fig. 3.

Again , plot the CO_2 concentrations by altitude and time.

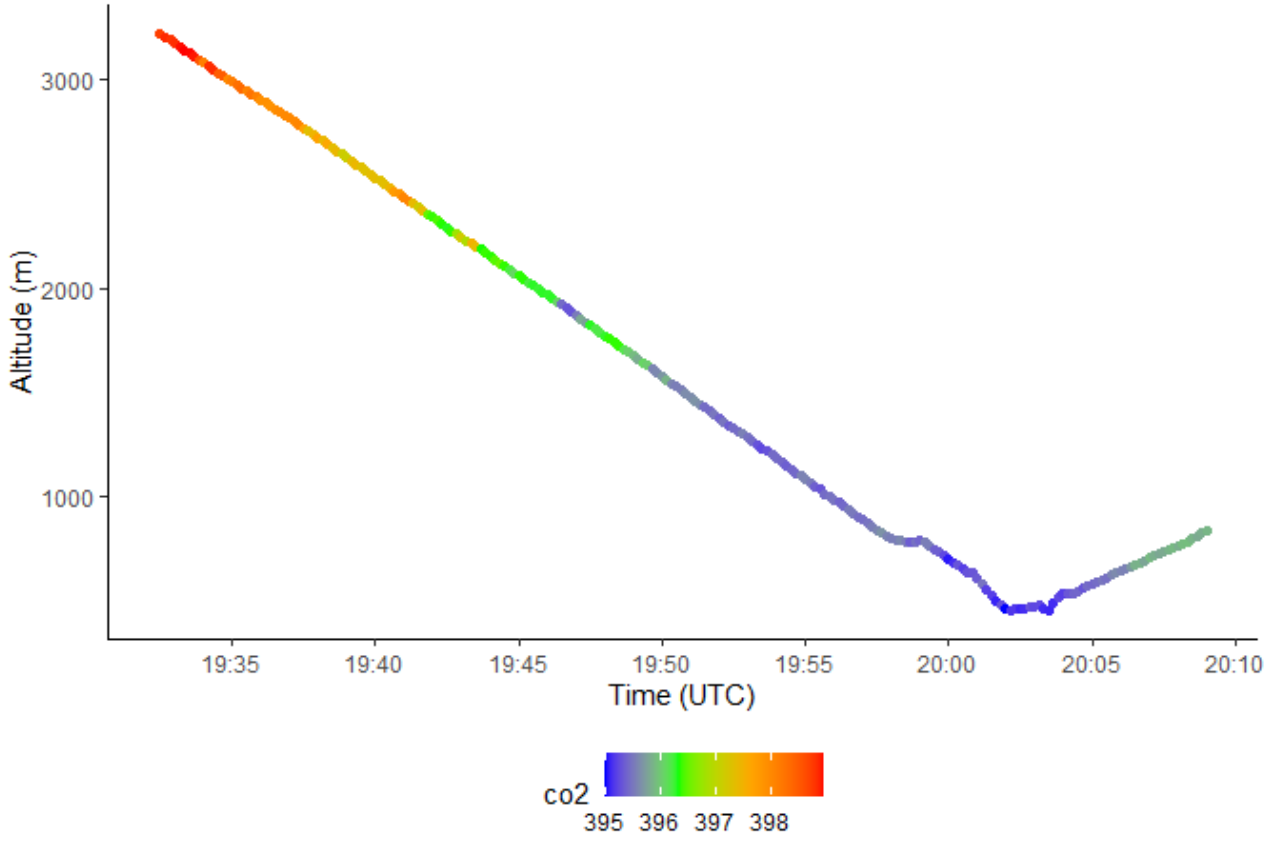

Figure 7. CO2 concentrations by altitude and time.

Figure 7 shows a clear gradient in CO_2 concentrations with altitude.

Data from a mixture of flights

Since flight lines were flown repeatedly, a spatial subset from the entire data set may include a confusing mixture of data from multiple flight lines and dates.

First, start over with the full data set, restore the dataframe from the backup copy created early in this exercise.

```
d <- d.bak
```

Then, create a subset from a portion of southeastern Alaska. Show the flight ID's, which correspond to dates.

```
lon.min <- -164.32   # western longitude
lon.max <- -156.51   # eastern longitude
lat.min <- 59.48   # southern latitude
lat.max <- 65.91   # northern latitude
d <- coord.filter(d, lon.min,lon.max,lat.min,lat.max)
# dim(d)  # 11578     7
unique(d$flight_id)     # show flights
```

```
##  [1] 20170430 20170501 20170608 20170609 20170713 20170714 20170823 20170824
##  [9] 20170826 20170827 20170917 20170918 20171023 20171031
```

This subset includes 13 flights, ranging in dates from April to October.

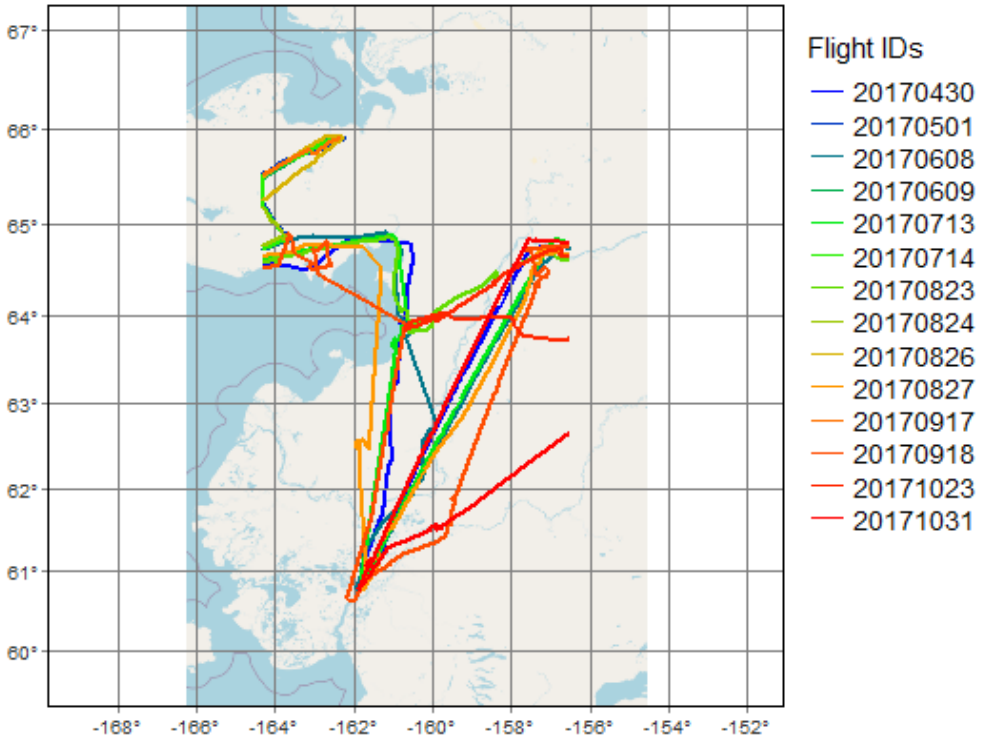

Figure 8. Path of flights from southeast Alaska.

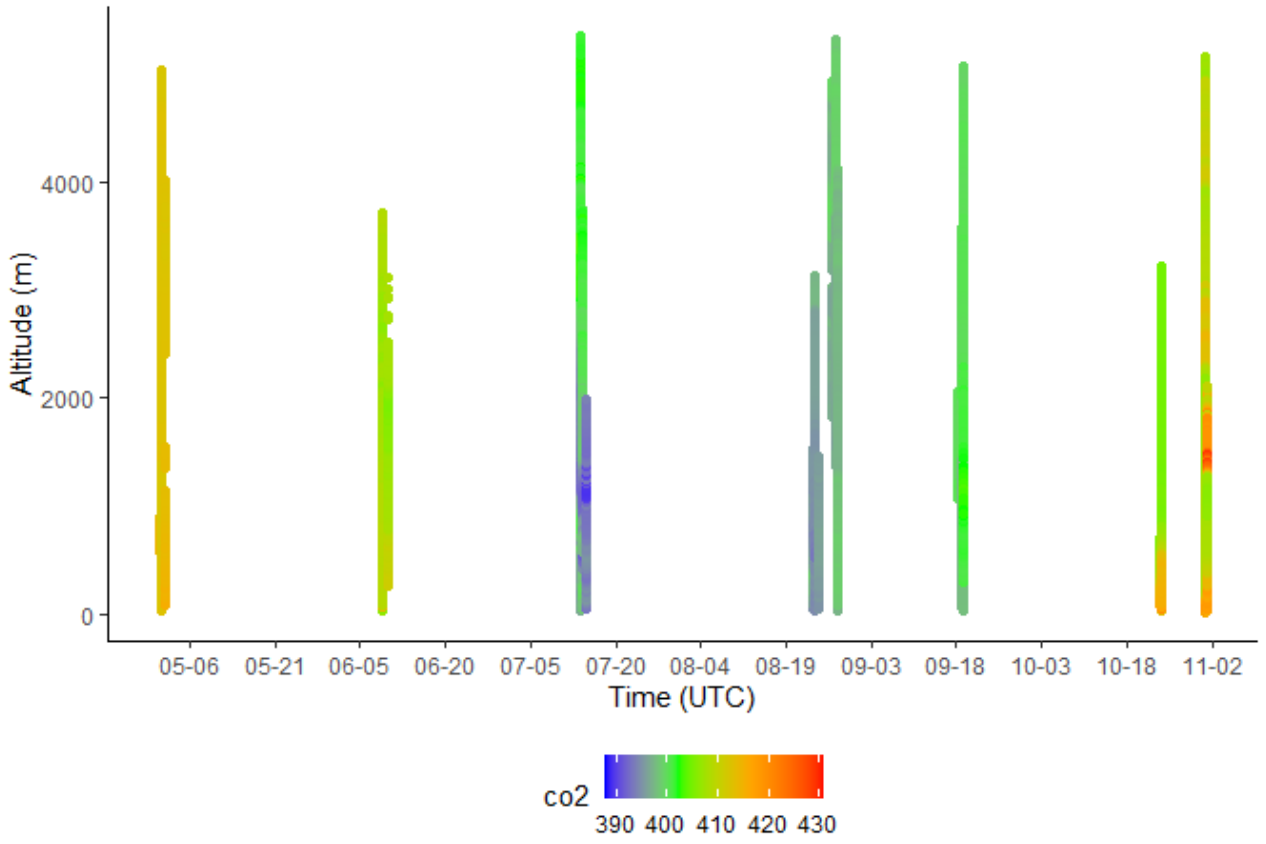

Figure 9. CO2 concentrations over southeast Alaska by altitude for multiple flights.

The plot of the CO_2 concentrations shows seasonal patterns by altitude, but the temporal details of individual flights are obscured. The CO_2 data for a specific flight or time period could be extracted by filtering by date (i.e., time.filter() function) or *flight_id* (flight.filter()) shown above.

Supplemental: Spatial subset by graphic

The code below allows the user to spatially subset the data by graphically by selecting the corners of the bounding box from a plot of geographic coordinates. This approach will work by running the code in the RStudio console or in the R GUI, but it does not work in R Markdown.

```
# Spatial select by picking bounding points from plot. ## Not run
plot(d$lon, d$lat)      # plot the flight path in a regular scatter plot
tmp <- locator()        # Click on corners of the area of interest (e.g., SW corner then NE corner),
                        # then click "Finish" in RStudio or right-click and "Stop" in GUI
tmp <- as.data.frame(tmp)     # passes locator() output to coord.filter function
xmin <- min(tmp$x); xmax <- max(tmp$x)
ymin <- min(tmp$y); ymax <- max(tmp$y)
d <- coord.filter(d, xmin,xmax,ymin,ymax)
```